ISSN 0956-9979 APRIL 2002

VIRUS BULLETIN

THE INTERNATIONAL PUBLICATION ON COMPUTER VIRUS PREVENTION, RECOGNITION AND REMOVAL

IN THIS ISSUE:

• **A dog's life:** Only last month Péter Ször estimated that we would need to wait a little longer for the appearance of 'the first known virus implemented in Microsoft C#'. The wait was shorter than expected, and this month Péter takes us through a detailed analysis of the first real C# virus, Gigabyte's W32.HLLP.Sharpei@mm. See p.4.

• **Flavour of the month.** A Unix theme pervades this issue of *VB*. While Marius van Oers looks at examples of Unix shell scripting malware on p.9, Aleksander Czarnowski shares some advice on how to secure *Linux* servers and desktops on p.12.

• **Comparatively new.** Continuing the Unix theme, Matt Ham embarks on his first comparative review for products on the *SuSE Linux* platform. Read all about it on p.16.

CONTENTS

COMMENT
You May Say I'm a Dreamer … 2

VIRUS PREVALENCE TABLE 3

NEWS
1. When is a Virus not a Virus? 3
2. Practise what they Preach 3
3. Suspicions Confirmed 3

VIRUS ANALYSES
1. Sharpei Behaviour 4
2. Juggling the Code 6

TECHNICAL FEATURE
Unix Shell Scripting Malware 9

TUTORIAL
Securing Linux Servers and Desktops 12

OPINON
The Easy Out, the Quick Fix, the Silver Bullet and the Big Idea 14

COMPARATIVE REVIEW
Making an Entrance: *SuSE Linux* 16

END NOTES AND NEWS 24

COMMENT

“ Borders are disappearing rapidly between viruses and what we call ‘malware’. ”

You May Say I’m a Dreamer …

John Lennon wrote ‘Imagine there’s no countries …’. Well, I don’t know about the real world, but in our anti-virus microcosm we are already there! The boundaries are blurred, and not only between different countries.

It comes as no surprise that modern virus infections spread like a bush fire across countries and continents. The Internet knows no borders, and the majority of computer users across the world run the same operating systems and use the same office productivity software. Someone in the US can be infected by a virus that originated in the Philippines, while someone in Australia produces a detection and cure for it. Virus writers have realized the power of team work and now get together in flocks on the Internet, hoping to develop something which will be incurable, impossible to beat and really fast-spreading. We have even observed virus localization services – for example, virus writers in Spanish-speaking countries translate viruses written by English-speaking authors into the native language – efficiency is the name of the game.

What can we do in response? I can see only one way to counter this: all anti-virus (and other security software) vendors need to work together, as well as with their users, even more closely than they do now. We are only strong if we are united to fight against these threats.

Borders are also disappearing rapidly between viruses and what we call ‘malware’. These days viruses may arrive on the back of an email worm, inside a Trojan Horse, sneak into the security hole of a popular Web server or pretend to be a JPEG of your grandma or anybody else whose image you might look at. I can just see a picture in my mind of a virus writer sitting in front of his computer, a smile on his face: ‘Which tool shall I use today?’. He could use any one of about a hundred different tools, SDKs, scripting languages, software packages and mutation engines. What is this? Is it a worm? Is it a vulnerability exploit? Is it a good old macro virus? Is it a Trojan? Is it a password stealer? It can be all of the above. We call it a virus with multiple infection vectors. Everybody remembers those – right?

It is not an easy task to get together AV software, intrusion detection, gateway content inspection, a firewall, VPN and strong authentication (in a form of PKI or biometrics) so that these packages do not conflict with each other. Very few vendors can present all of the above plus other bits and pieces which would help security administrators to manage that software across tens (or hundreds) of thousands of desktops. Regardless of whether you chose one vendor (to take advantage of discounts and to deal with the same support guy) or a selection, you would still need all of the above to protect your networks if you consider your company data and your employees’ time to be worth anything.

And yet another border is becoming very blurred. This is a border between malware and commercially produced software. (I don’t want to mention names, as it would represent free advertising for these vendors). How many times (and I think I am speaking for all AV vendors) have our customers requested (sometimes in a very forceful manner) detection of a particular piece of software which is sold commercially? The majority of our users don’t want any software which can be installed or run silently on their networks and which can give some third party (either on the same network or anywhere on the Internet) control over their computers.

Legally, we have a problem here, as manufacturers whose software has become classified as malware are threatening to sue the AV vendors – they, seemingly, don’t understand or don’t *want* to understand why the majority of users want to know when such software is anywhere around their networks. For this one I don’t have a good answer and I would love to hear feedback from the users and from fellow AV companies on this matter.

By the way, sometimes I wish I could write songs as John did. Well, dream on!

Dr Eugene Dozortsev, Assist. Vice President R&D, eTrust, Computer Associates, Australia

NEWS

When is a Virus not a Virus?

The last few weeks have seen a plethora of incorrectly coded worms. Well, two at least.

Both W32/FBound.C@mm and W32/Gibe@mm have caused a few problems in the detection process. Both worms contain errors that have caused some of the files mailed as attachments to be non-executable crud. Traditionally, anti-virus products have detected viruses and have lately made the concession of adding detection for Trojans and other malware species. Although from time to time non-executable files turn up in collections, there has (until now) been no need for anti-virus vendors to include detection for them in their products. However, the situation is a little different where these worms are concerned, since customers have been receiving files which *appear* to be viruses. You can expect various detection solutions to be included in AV products within the next six months ▮

Practise what they Preach

An informal survey of RSA Conference 2002 attendees has found that a shameful 91 per cent of them break their own company security policies.

The 'Know Your Enemy Security Survey', which was conducted at the security event in February by email security appliance developer *CipherTrust*, revealed that security policies are violated at every level within corporations. In fact, 25 per cent of respondents identified someone at the CEO/CIO level as having been the person responsible for launching CodeRed and/or Nimda within their organization. More of the survey's findings can be found on *CipherTrust*'s Web site (http://www.ciphertrust.com/) though, somewhat disappointingly, there is no naming and shaming ▮

Suspicions Confirmed

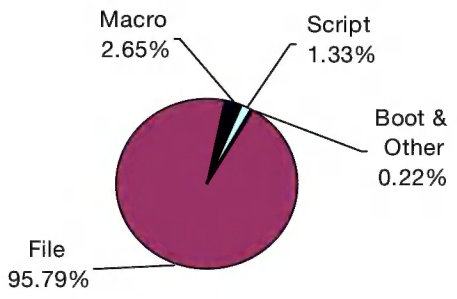
The *ICSA Labs 7th Annual Virus Prevalence Survey* has confirmed that the rate of malicious code infection continues to rise. The survey, which gathered data from 300 companies and government agencies, aims to track trends in malicious code and the way in which organizations attempt to prevent or combat it. The survey identified an increase in the number of multiple vector threats and Internet host-based threats and highlighted a number of factors that contribute to the rising infection rates – including an increased use of multiple email programs, new replication vectors and expanded forms of connectivity. Dr Peter Tippet, chief technologist at *TruSecure Corporation*, called for AV vendors to provide more heuristic tools. The survey can be downloaded from http://www.trusecure.com/ ▮

Prevalence Table – February 2002

Virus	Type	Incidents	Reports
Win32/SirCam	File	1427	28.51%
Win32/BadTrans	File	1069	21.35%
Win32/Klez	File	953	19.04%
Win32/Magistr	File	561	11.21%
Win32/Myparty	File	247	4.93%
Win32/Hybris	File	187	3.74%
Win32/Nimda	File	92	1.84%
Win32/MTX	File	48	0.96%
Laroux	Macro	42	0.84%
Haptime	Script	35	0.70%
Win32/Aliz	File	34	0.68%
Win32/Goner	File	31	0.62%
Kak	Script	25	0.50%
VCX	Macro	21	0.42%
VBSWG	Script	20	0.40%
Win32/Gokar	File	14	0.28%
Ethan	Macro	11	0.22%
Divi	Macro	10	0.20%
LoveLetter	Script	10	0.20%
Win32/QAZ	File	10	0.20%
Bablas	Macro	8	0.16%
Win32/Ska	File	8	0.16%
Win95/CIH	File	8	0.16%
Marker	Macro	7	0.14%
Others [1]		128	2.56%
Total		5006	100%

[1] The Prevalence Table includes a total of 128 reports across 59 further viruses. Readers are reminded that a complete listing is posted at http://www.virusbtn.com/Prevalence/.

Distribution of virus types in reports



VIRUS ANALYSIS 1

Sharpei Behaviour

Péter Ször
Symantec Security Response, USA

In a recent article in *Virus Bulletin* (see *VB* March 2002, p.6) I described the internal details of W32/Donut. In my article I pointed out that, contrary to some reports, W32/Donut was *not* 'the first C# virus'. In fact, Donut contained no functional C# code.

Shortly after the appearance of Donut, however, we received another virus – W32.HLLP.Sharpei@mm, written by Gigabyte. Unlike Donut, this virus does contain functional C# code and is the first *real* C# virus.

Infection Trivia

Having stated 'it is not a trivial task to write a C# virus', I was asked recently by a member of AVIEN (Anti-Virus Information Exchange Network) whether the virus writer had beaten my expectations. Indeed, I had not expected the first C# virus to appear so quickly.

However, I still believe it to be non-trivial to infect *.NET* files with a piece of code written in C# that will infect other *.NET* files at their C# entry point. W32/Donut demonstrated how simple it can be for a virus to infect a *.NET* file at its 32-bit entry point.

Infection techniques that do not pay attention to the *.NET* file format can be considered trivial. The HLLP method (High Level Language Parasitic) is an example of such an easy technique. This simple direct action infection is carried out by a C# component of the virus code.

The virus works as a mass-mailer even when the *.NET* framework is not installed. This makes it a current problem rather than a future threat, although the VBS script that is used for the mass mailing can be detected by today's heuristics as well as script blocking techniques.

You Have Mail

W32.HLLP.Sharpei@mm arrives as an email message with the subject '*Important: Windows update*'. The body of the email contains the text: '*Hey, at work we are applying this update because it makes Windows over 50% faster and more secure. I thought I should forward it as you may like it.*' The email arrives with an attachment named 'Ms02-010.exe'.

The actual format of the virus is a regular 32-bit PE file. I imagine that the assembly portion would have been created by the NGVCK kit and then altered with new code. The size of the first generation sample is 12,288 bytes. This executable contains a VBS script as well as a *.NET* PE file which is written in C# and contains MSIL code.

When the attachment is executed, the virus makes a copy of itself as C:\Ms02-010.exe. It drops a VBS file with the name 'Sharp.vbs', which performs the mass-mailing routine, sending the message as described.

Finally, Sharp.vbs deletes itself. Once the messages have been sent successfully, they are deleted from the *Outlook* Sent folder. As a result, you will not see any evidence of the messages in *Outlook*. This is an attempt by the virus to hide its activity.

If Mscoree.dll is found in the \System folder, the virus creates 'Cs.exe' in the \Windows folder, then executes it. Sharpei makes the assumption that this library is present only when the *Microsoft .NET* Framework is installed. Cs.exe is a 7680-byte *.NET* executable that is written in C# and runs only in the *.NET* Framework.

Finally, Ms02-010.exe creates the HKLM\Software\Sharp key in the registry and sets this to point to the executed infected file. Thus, in the first execution, this might point to 'C:\mymail\ms02-010.exe'. This string is used later on as a reference from Cs.exe (the *.NET* component) to the executed attachment or infected application.

Cs.exe in Action

This is the portion of the code written in C# that implements buggy direct action prepender virus logic. It works properly for first-generation infections.

(In order to understand this code it is necessary to have IDA or the ildasm.exe utility to produce an output containing MSIL code. Reading MSIL will be a new skill for virus researchers to learn – at first glance the stack machine code is a little confusing to someone who reads assembly language, which uses registers for most tasks.)

First, the special '.cctor()' constructor function sets a local variable to the 'Sharp' key in the registry, referencing the executed infected application. Initially, this points to the 12,288-byte attachment, but later on it will point to the infected file with the host application appended to the virus. Basically this is a confusing way to pass a parameter to Cs.exe.

Next, the .entrypoint method takes over and uses System.Environment::GetFolderPath to determine the path of the Startup folder. Then a short sharp.vbs file is created in the Startup folder with a message box which states 'You're infected with W32.HLLP.Sharp, written in C#, by Gigabyte/Metaphase'. The message box, as shown in the picture above, will be displayed when the machine is

rebooted. Then the virus figures out the path to the Windows and the Program Files folders respectively.

FileSearch

Next, Sharpei calls its own 'FileSearch' function four times in order to look into three subdirectories of the Program Files folder and the Windows directory. Sharpei searches in these folders for files with the '.exe' extension.

FileSearch is the infection function. First it reads the checksum field of the MZ header (0x12 file offset) to see if the marker 'g' is present. This marker is placed in the first generation sample.

If the marker is found the virus attempts to delete the file 'hostcopy.exe' (which is used during the infection) and searches for the next file to infect.

If the marker is not found the virus clears the attributes of the file and makes a copy of it as 'hostcopy.exe'. It uses the System.IO.File::Copy function with the third parameter set to 1, which means that the target will be overwritten should it exist.

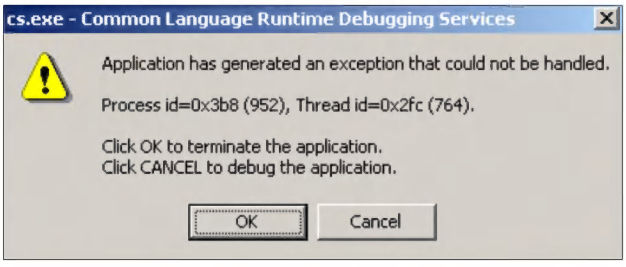
Then the virus uses the default value of the 'Sharp' key to copy the referenced file over the victim file. Finally, 'hostcopy.exe' is appended to the file.

After every infection 'hostcopy.exe' is deleted (it is only a temporary file). I should note that during test infection of the most recent version of the *.NET* Framework we experienced a number of error messages displayed by the Framework, as shown below.

When all four directories have been searched for infected files the virus attempts to check whether there is a host application within the executed application. The virus reads the full content of the host from position 0x3000 (12288) of the virus and moves that content to a new temporary file of its own called 'temp.exe'.

Routine Bugs

This routine suffers from a couple of bugs. First, the virus checks whether the executed file is 'MS02-010.exe' and if this is the case it attempts to delete 'temp.exe'. However, finally it will attempt to execute 'temp.exe' even if it is not there. It uses System.Diagnostics.Process.Start to execute the host temp file.



Whenever a pre-infected application is executed the new target will have a copy of the virus with a previous host appended to it and the new host appended again and so on. Thus each generation will become longer and longer and in many cases Sharpei will not be able to execute the proper host application.

Unfortunately this also means that it is impossible to restore the proper host application with AV repair. This is because the virus pays attention only to the file extension, rather than the file format.

It would be difficult to find the last host in the infected file in all cases. Thus I would recommend deletion for removal of the virus rather than repair. Otherwise the system would contain repaired files with confusing host content.

Dog Tag

Was it wise to name the virus after a dog breed? Well, it seemed a logical name to me. Gigabyte, the author of the virus, wanted to it to be called 'Sharp'. Originally, the Chinese Shar-Pei breeders cultivated the excess folds of skin on the Shar-Pei dog to give the breed an advantage in dogfights. In this virus the C# (C sharp) code was placed under a 'skin' (the 32-bit and VBS layer) to give the virus an advantage.

Conclusion

W32.HLLP.Sharpei@mm demonstrates that it is relatively simple to write a prepender virus in C#.

Administrators need to make themselves familiar with the *.NET* Framework security settings and should not leave the default configurations installed. Since the *.NET* security config files are found in the Windows folder it is important to make sure that the OS level security is used properly as well, otherwise viruses might change the settings one way or another.

Name: W32.HLLP.Sharpei@mm	
Alias:	W32.HLLP.Sharp.
Type:	Uses VBS file to mass mail, direct action virus under *.NET* Framework.
Size:	12,288 bytes.
Payload:	Displays a message box on system start.
Repair:	Delete infected files and restore from backup.

VIRUS ANALYSIS 2

Juggling the Code

Gabor Szappanos
VirusBuster, Hungary

Having written a couple of analyses of 'polymorphic' macro viruses, I was left with a certain uncomfortable feeling that I couldn't shake off – I could not wholeheartedly call all of those samples polymorphic.

The polymorphic nature of most of the samples was a result of the fact that macro viruses contain non-compiled source code. The vast majority of the so-called polymorphic macro viruses are simple variable-name changers.

WM97/Jug.A

However, I feel somewhat more comfortable now that I can present WM97/Jug.A. This virus can be considered polymorphic by even the strictest definitions.

Most of WM97/Jug.A's virus code is encrypted with a variable length encryption table that is changed with every infection. The encrypted virus body is itself variable, with random junk strings appended to the end of the code line and inserted into random locations within the code. Even the decryptor is gathered randomly by the polymorphic engine.

All in all, this virus is the most complex polymorphic macro virus I have encountered.

General Operation

WM97/Jug.A activates whenever an infected document or workbook is opened. The virus and the polymorphic engine are capable of working from within both *Word* and *Excel*, creating an *Excel*-specific decryptor in an *Excel* workbook, or a *Word*-specific decryptor when run in *Word*.

However, the virus is not an *Office* cross-application infector, as it does not have the capability to jump from one application to the other: if the virus is running from a *Word* document, it will only infect other *Word* documents and vice versa.

As the *Excel* components of the virus show some instability and have a tendency to crash *Excel*, this analysis will concentrate on operation within *Word*, but will mention the *Excel*-specific parts also.

Upon activation from a *Word* document, the virus creates a new empty document, then decodes its coded main virus body (which is stored as a comment block), inserts it into the new document, then closes the new document without saving it.

This action will fire up the Document_Close (or Workbook_Close in the case of *Excel*) event handler in the temporary document, which will infect the global template or further documents, and finally closes the document losing all changes – including the decrypted virus code.

Activation in Detail

When the virus is activated, it first disables the *Word* and *Excel 2000* virus protection by setting it to the lowest level that allows the execution of all macros. Moreover, the virus disables the Visual Basic toolbar and the Macro|Security, Window|Unhide, Tools|Macro, Tools|Customize, Tools|"Templates and Add-Ins..." command bar buttons, not allowing access to the Visual Basic editor environment through the menu (though the virus does not disable the Alt+F1 shortcut).

Next the virus either infects the open documents, workbooks and NORMAL.DOT, or it infects all documents (or workbooks) in a collection of directories – with equal probability. Then, if the current date is 18 May, the virus will replace the current selected text (or inserts at the cursor position, if nothing is selected) with the following text:

```
========
Love has torn us apart...
XXX
 years..
========
```

where XXX is the number of years since 1980.

Then WM97/Jug.A picks a random number, and depending on its value, with a one in 11 chance the virus will infect the host application using Sendkeys.

The virus will send key combinations that insert the mutated virus code, already generated and stored in the file C:\TEST.TXT, into the last code pane window in the VBE editor – whichever object it belongs to. To be on the safe side, almost each keypress is transmitted three times.

The virus activates one of four possible payloads, each with a one in 11 chance:

- In a primitive DoS attack, the virus spawns five shells that ping ftp.nai.com in infinite loops.
- The virus registers the *Minesweeper* game to start automatically at boot.
- The virus opens http://www.nephilim.com/ in an explorer window (this contains an empty Web page).
- The virus attempts to download Back Orifice 2000 from its 'official' Web site, http://www.bo2k.com/ (the site no longer exists).

If it is the eighteenth day of the month, or the value of the current minutes happens to be smaller than the value of the current seconds, the virus will mass-mail itself using *Outlook* – again with a one in 11 chance.

The subject and the content of the outgoing mail is selected randomly from a list of 15 possible subject body pairs.

The possible subjects are:

'Here's {Document.Name}'

'Hope this is what you wanted...'

'Hey John, check this out [private]'

'Check out these sitez! {personal}'

'The report you requested'

'Check out these babes!'

'New computer question...'

'That report...'

'...'

'IMPORTANT: virus alert'

'Latest round of Bill Gates jokes...'

'<attachment>'

'Microsoft Office 2000 code crack!'

'Pictures'

'New Windows Internet hack found!'

While the corresponding message bodies are (in the same order):

'Hope this helps. {UserName}'

'I've attached {Document.Name} Let me know if there's a problem.'

'(Make sure you don't show anyone else)'

'Don't stay up TOO late checking these out :-)'

'If this isn't the right one, let me know. {UserName}'

'You know the score, passwords to the latest batch of sites enclosed (courtesy of FoG) .(Check out the girl on the entrance to site number 3...)'

'Does this look right? New PIII for $600? What's missing from it?'

'Here's {Document.Name} This was what you wanted? Call me if it isn't...'

'Free porn site passwords!'

'Do we need to be worried? {UserName}'

'Joke 4 nearly gave me a heart attack, I laughed so hard!!'

'Here's "{Document.Name} , I hope this was the one you wanted.'

'I've attached the serial number, and FTP site/password. Don't want to be caught out by Y2K (or spend $1300 either!)'

'Here are those pictures of me you wanted — enjoy ;). I've put them in Word format, the password is 'LUVMYBODY'.'

'Check this out, do you think it's for real?'

The virus will send messages to all the email addresses in the Contacts folder. If the current number of seconds is less than 15, separate messages will be sent to each address, while if the number of seconds is higher than 15 a single email will be sent, with all the addresses together in the recipient list.

Infection Procedure

The virus has two main infection methods; one of these is picked at random and executed upon activation.

The first method of infection inserts the mutated code into *Word*'s global template (if the code is run from within *Word*). If the code is executed from within *Excel*, the virus creates a new workbook and inserts the mutated code there. Following this, WM/97Jug.A infects all open *Word* documents or all open and unsaved *Excel* workbooks.

The second infection method searches through the hard disk looking in specific directories for possible targets.

The directories are:

- The current folder
- The default folder for storing documents (usually 'My Documents')
- The folder where the active document or workbook resides
- The root directory of the active document or workbook folder

The virus keeps track of which directories have been infected already, thus avoiding the double processing overhead in case any of the directories from the above list are identical.

Depending on the host application, all *.DOC files (*Word*), all *.XLS files (*Excel*) or all *.VBS files (script) are enumerated. The last option is probably for further development in the future, as no methods exist currently for handling files other than *Word* documents and *Excel* workbooks.

Every file that is found is infected by a mutated copy of the virus.

Polymorphic Engine Exposed

Certainly the most complex part of the virus is the polymorphic engine. It consists of two consecutive procedures; the first inserts random comments into the main virus body, and then encodes it, while the second part generates the decryptor.

First the engine attempts to find the encryption table stored in the decryptor, which is recognized as any string coming after Asc(Mid(" in the target code module. If such a string is found, the virus attempts to decode the first line of the target code module with this key.

If the decoded string contains the string 'Nephalim v0', then the document is considered to be already infected with some version of Jug, and is therefore left alone.

If the document is not found to be infected, the virus creates a new encryption key (consisting of six to 35 characters from the ASCII range 65…122), then a random suffix is added with a one in three chance after each virus code line in order to change the line length.

The postfix consists of :NPR and six to 35 random characters. Note that the postfix is not separated from the virus code with a comment – the code line in this form would not be executable. However, the decryptor recognizes these postfixes by the :NPR string, and removes them from the decrypted virus code.

After each code line the virus will insert a random junk line with a one in three chance. The random line will start with 'NP and six to 115 random characters.

These random comments serve the purpose of ensuring that the encrypted body does not have a fixed number of lines and that the encrypted lines do not have fixed lengths; these parameters vary with each infection.

Encoding and Decoding

Once the virus body is ready with all comments inserted, the virus encodes it with the newly generated encryption table.

The coding is simple: each character is shifted with the ASCII value of the encryption table character (for each character at a position over the encryption table length, the first character of the table will be used), with special care taken that the coded character's ASCII value would be between 32 and 130.

The final step is the creation of the random decoder. The decryptor also utilizes variable-name changing. Each variable is one byte long and randomly selected from the 'A' … 'Z' range. Special care is taken to ensure that variable names are different.

There are two crucial object variables, namely *Z.VBProject.VBComponents.Item(Y).CodeModule* and *VBProject.VBComponents.Item(1).CodeModule* (where Z and Y are each one of the random variable names generated). These are both cut into two pieces along one of the '.' dereferences, and stored in two variables each.

The first of the two is stored in a variable, and all further references will appear as relative to that variable object – for example, R will be *'Z.VBProject.VBComponents'*, and all subsequent references to this will appear as R. *Item(Y).CodeModule*.

Thus, it is not easy to lay a scan string on these references for detection.

Furthermore, the numerical constants in the code (virus code line count, character range borders) are each generated as a sum of two numbers.

The virus line count is 130, therefore a random number between 0 and 129 is generated (say 35), and the line count is referenced later in the generated decoder as, for example, 35+95.

Next the code lines are generated one by one. For those lines that can be merged together, a line connector is selected at random. This may be ':' (which means that the line is joined with the next one), a line feed, two line feeds, or a random comment and a line feed.

For those lines that cannot be joined with the next line (e.g. the Else in an If statement), a random line terminator is picked. This can be one to three consecutive line feeds.

After the decryptor, almost empty Document_Open and Workbook_Open procedures are generated. These procedures are almost empty since they contain only a call to the virus decryptor. The declaration of these procedures is also generated randomly; it can be '*Private Sub*', '*Public Sub*' or plain '*Sub*'.

Conclusions

Clearly WM97/Jug.A was written by a disciplined programmer. The code and the procedures are carefully written, the variables are consistently named with type prefixes.

The virus proves that, contrary to common belief, macro viruses are not history, there are still serious virus writers working on new creations. WM97/Jug.A comes very close to a level of polymorphism which is undetectable by the methods of macro virus identification that are used currently.

WM97/Jug.A	
Aliases:	Jugular, Nephalim.
Type:	Polymorph encrypted *Word/Excel* macro virus.
Payload:	DoS attack against ftp.nai.com upon each activation with a 1:11 chance.
Self-recognition:	If the target contains the encrypted string 'Nephalim v0', it is left alone.
Removal:	Delete the content of the infected code module, or use a virus scanner.

TECHNICAL FEATURE

Unix Shell Scripting Malware

Marius van Oers
McAfee AVERT, The Netherlands

Unix/*Linux* binary malware can be very dependent upon distribution flavour and kernel version. Furthermore, the use of binary files as a starting point for virus infection may not always be very successful – starting off with a coredump will result in a rapid failure.

In the past we have seen worms (for example Linux/Adore) make use of a combination of ELF binary files and scripts. Usually scripts are independent of distribution flavour and kernel version, and most are likely to have few problems running on the target machine.

For some worm packages, scripts act as the 'fire-starters' on the target PC. The scripts may execute directly, or may call other script files and binaries. Sometimes local files are replaced by compromised ones that are included in the worm package.

So what are the possibilities in the Unix world for malicious code using scripting?

Unix Scripting

There are a number of different Unix/*Linux* distributions, the majority of which support scripting. Similarly, there are a number of different forms of scripting.

Javascript is supported on both *Windows* and on most Unix/*Linux* systems. Therefore, the creation of Javascript malware that will work in both operating system environments is technically possible, and it should be relatively easy to accomplish.

The binary infector W32/Lindose was a 32-bit PE *Windows*-based infector that searched the system for binary ELF files to infect, however Lindose does not operate in the opposite direction (i.e Unix to *Windows*). Technically, this should have been achievable – consider, for example, that emulator programs exist on Unix systems to run Win32 code. However, a considerable level of technical expertise would be needed to achieve this and, more importantly, it would be a significantly time-consuming process. It would be both quicker and easier to write a Javascript virus that can run natively in both the *Windows* and Unix environments.

The powerful Perl scripting is supported on a lot of Unix systems, either installed directly or using an add-on package. A sample file might be called 'runme.pl'.

Unix shell scripting is very powerful too; it may control program configuration and start/kill services. Unix shell scripting has many flavours, for example Bourne (sh), Bourne Again (Bash), Korn, C and Tops C shell scripting. Also it is possible to create a completely new shell interpreter. However, the most common is the Bourne Again shell scripting, using the '/bin/sh' interpreter. A sample file might be called 'runme.sh'.

A virus writer making the assumption that Bourne Again is the default shell interpreter runs the risk, should this not be the case, of the virus producing errors and crashing. A simple way to avoid this situation is to insert a '#!/bin/sh' line at the start of the file.

On *Linux* systems '#!/bin/sh' will act as a redirect to Bash, but on other Unix systems there are differences between sh and Bash. An alternative shell interpreter can be specified, for example using '#!/bin/csh'. On *Solaris* systems the Korn shell, ksh, is used widely.

Now let's take a closer look at Bourne shell scripting and the malware making use of it.

Unix Shell Malware

Creating malware using shell scripting is relatively easy. Simple viruses may be very short, consisting of only a few lines, and even less code is needed to construct a Trojan.

Another aspect is that, unlike binaries, the Bourne shell scripts will (usually) work on a large number of different Unix flavours (or will do so with a few very minor modifications).

By examining some samples that were distributed in the latest publication of a well-known virus-writing group, we can take a look at what possibilities and techniques exist for shell viruses.

Determining Which Files to Infect

With the support of 'if-then-else' and 'for-do' loops it is easy to create viruses that search files for suitable targets. The search can be carried out both in the current directory and in others, using directory walking loops.

Suppose we have a simple Bourne shell virus; without filtering the viral shell code could be added to binary files. So, in order to prevent unexpected results, proper filtering is required.

Grep

Usually Bourne shell scripts start with a reference to the interpreter, '/bin/sh', in the file header. So a quick check for files that start with '#!/bin/sh' would provide a good subset of initial target files for infection.

This search is possible using the 'grep' command. For example:

```
'grep -s #!/bin/sh $targetfiles'
```

In this case the '–s' option is used to suppress any error messages.

Head

Instead of examining the complete file, the head of the file can provide useful information for faster filtering.

The 'head' command returns information on the beginning of a file. For example, 'head –20 $file' returns the first 20 lines of the file, while 'head –c20 $file' returns the first 20 characters of the file.

File

Using the command 'file' it is possible to determine whether the filetype of a target file is of Bourne shell format. However, this technique is rarely used; it is not a perfect technique, as it reads file headers to determine the file type.

In some cases, for example with .sh scripts, it is not necessary for shell scripts to have lines such as ' #!/bin/sh' at the beginning of the file. Although this command interpreter line is encountered frequently, it is not mandatory. Files without the expected command interpreter line could be judged by 'file' to be regular ASCII files rather than shell script files.

Find

Unix systems have a wide range of protection techniques, so, in addition to this file checking, a virus should investigate the target file's permissions – for example, determine whether these are set to read (-r-) , write (-w-) and/or executable (-x-).

Some viruses walk through directories/folder trees but upon infection fail to check whether the target is a file or directory, which may result in crashes.

The 'find' command can be used to search for specific target files. And, not only can 'find' filter on files with specific attributes (-r -w -x etc.), but it can also execute a command on the target files that are found. However, making use of 'find' may result in a noticeable decrease in the speed of the system.

To prevent an early discovery by a user, it is possible to launch processes in the background, using the '&' shell script symbol.

Temp Files

To avoid speed reduction, script viruses may create temporary files. The viral code can be copied to these and any time-consuming routines can be run from there in the background. This way the process remains transparent to the user – there is no obvious decrease in the speed of the host application.

Another reason for making use of temporary files is to differentiate between the pure viral body and those files that are being infected. Some viruses copy the target file to the temp folder, modify it, and write back, replacing the now infected target file.

If there are errors, or corrupted files, it's easier to hide them by using a central, temporary, location than it is when working directly in the target file directory. Although error messages can be caught and redirected to null.

Bash allows redirection of the standard output to other files, by making use of '>'. Redirecting standard error output is possible also by using the '2>' symbol – for example, '2>/dev/null' (for sh and similar shells).

So a specific search selector could resemble the following:

```
… if [ "$(head -c9 $F 2>/dev/null)" = "#!/bin/sh"] …
```

This translates as: find files ($F), examine the first nine characters of the file and verify whether it is #!/bin/sh (the Bourne shell command line interpreter), while redirecting error messages to null.

To mark an infection, a simple, yet specific, marking can be used. Searching for the presence of an infection marker can also be done by using 'grep' or a similar technique as described above.

Infection Spectra

Unix shell viruses can:

- Prepend the viral code. Prepending viral code is pretty easy to do, the viral code is always executed. However, the drawback is that prepending viruses are easy to spot.
- Append the viral code. A simple tail –n 25 $0 >> target file will append 25 lines of the viral code to the target file. However, appending viral code might not always be called. If there's an error in the 'host' program, or it terminates with an exit code, the appended viral code won't be called. Usually script file code is executed from the beginning to the end of the file though, so both the host and the viral code will be called by the interpreter.
- Overwrite the target file with the viral code. Overwriting target files is, as such, already a rudimentary method but without proper file-type checking it may replace ELF-type binary files with ASCII-type script code.
- Insert the viral code somewhere inside the target file. This is more difficult for a user to detect, and might result in errors if certain host program code can't

complete (due, for example, to a crash by the inserted viral code).

- Create companion files. Usually the original file becomes hidden, the viral code takes the original host file name, while maintaining the same file attributes as the original host.
- Insert a call in the target file, in so doing leaving the real viral code in another file.
- Have encrypted/polymorphic code. Encrypting files can be easy. A simple ASCII-HEX conversion would make the code unreadable for most end users. ASCII to Hex conversions are possible using the 'printf' (\x123) command. Creating 'polymorphic' script viruses is pretty easy to do. One can insert random comments, or change variable names. Usually the random generator is supported, but other variables such as current date can be used as well.
- Use Sendmail. So far the use of Sendmail in Unix shell scripting malware is limited. In fact, this is quite remarkable as Unix systems can control mail programs often and are easy to call. A single line of code could call the program. Luckily, no successful Unix shell scripting mass-mailing worm has yet been encountered in the wild.
- Use another shell interpreter and recompile its code on a current system to avoid the incompatibility between its binaries and the operating system (see Unix/Cliph).
- Exploit security vulnerabilities in order to compromise the root account (see Unix/Cliph).

Sample 1: Unix/Zerto

This sample (filename elfo.sh) was included in a recent publication by a popular viral group.

The elfo.sh file starts with its identifier, marker (#;P) and Bourne shell interpreter, #!/bin/sh. Then the code performs a search on suitable files to check for the infection marker using:

```
[ -f $F ] && [ -x $F ] && [ "$(head -c3 $F)" != "#;P" ]
```

It searches for (-f) present, normal files that are flagged as (-x) executable and whose first three characters are not '#;P', thus checking that the specific file hasn't been infected already.

The virus takes the prepended viral code, the top 27 lines of the file, and copies the code to a tmp file. It then marks the file as executable/runnable and starts it.

Possible errors are redirected to null, thus hiding any error messages. The infector process runs in the background, this is mainly for speed considerations.

Host files are copied to the tmp directory and infected. Then the virus moves the tmp file back to the (now infected) host, and deletes the tmp file.

At this stage the viral script code should be prepended to an executable file, for example a shell script or ELF binary file.

However, when the virus sample that was provided was run on a *Linux RedHat 7.0* test system, a number of errors were produced.

Sample 2: Unix/Cliph

This backdoor sample (filename smlix.sh) came from a virus collection site and was discovered in August 2001. It is a *Linux* kernel 2.2.X (X<=15) & sendmail <= 8.10.1 local root exploit.

The malicious code starts with a reference to the shell command line interpreter '#!/bin/sh'. However, the code uses another shell interpreter in addition, namely tcsh: SHELL=/bin/tcsh.

The virus creates an anti-noexec library called 'capdrop.c' and attempts to compile it into a binary called 'capdrop.so'. Local recompilation is used to prevent problems that could be encountered when running binaries on different *Linux* distributions.

However, when the virus sample that was provided was run on a *Linux* test system, a number of errors were produced.

General Issues with Infecting

Creating a shell script virus sounds straightforward, but in practice a lot can go wrong during execution.

Apart from access rights, the viral file itself can sometimes be tricky to run successfully. One of the items that is overlooked sometimes is the exact end of the file. Without the new line symbol some viruses may fail to execute properly, resulting in errors.

Conclusion

Unix shell script viruses are relatively easy to create, yet powerful enough to create big problems.

Power users are likely to be alerted to malicious changes to their systems pretty quickly, but as more novice users migrate to popular *Linux* distributions such as *RedHat*, shell script malware may go unnoticed. More importantly, the novice users provide the less secure environments for malware to exploit.

At this stage, Unix shell script malware as such is more targeted at the specific machine – currently it doesn't spread its code to other machines natively. So far, it couldn't survive on its own.

Unix viral packages that have been successful have consisted of both binaries and scripts. However, there is no technical reason why Unix shell script malware cannot be successful in the future – it is a matter of proper coding combined with suitable (less secure) environments.

TUTORIAL

Securing Linux Servers and Desktops

Aleksander Czarnowski
AVET Information and Network Security, Poland

At what point can one say that an operating platform is popular? From a security point of view, we could say that a platform is popular when at least three viruses exist that target that platform.

If you count Trojan horses and rootkits, *Linux* (as a Unix derivative) crossed this line a long time ago. So it is high time that we designed and implemented some anti-virus schemes for *Linux* hosts. While there are hundreds of papers describing the process of securing or strengthening *Linux* (and, yes, this is another indication that *Linux* is popular), very few of them consider security from the perspective of an institution's anti-virus policy. The aim of this article is to shed some light on this subject.

Problems

The first problem is the fact that *Linux* (or rather GNU/*Linux*) is just the name of the kernel. What real users get on installation CDs or from the Web is a distribution that is a compilation of different applications (in binary form) from different sources with a ready-to-use *Linux* kernel.

The second problem lies in the kernel and kernel extensions. The *Linux* kernel is developed and updated constantly and several versions have introduced some interesting security features and vulnerabilities.

There are also security extensions for the *Linux* kernel, such as OpenWall (http://www.openwall.com/linux/) or LIDS (http://www.lids.org/). These can be applied only to specific kernel versions. Even if you find two distributions with the same kernel version, you may still have very different installations.

The next problem is the fact that you should install any application by compiling its source (but only after MD5 checksum has been verified positively). However, in the real world this is not very common, especially when RPM and *Debian* packages are widely available. While the use of packages aids the installation of applications in a system, for security reasons it is not a recommended option. This is especially true for server applications.

Additional Considerations

As you might expect, there are advantages and disadvantages for every type of *Linux* distribution.

For example, *SuSE* distribution is huge. There is a good chance that everything you will ever need is provided on the distribution CDs. From a security perspective this can be a drawback because it must be very hard for the developer to maintain one security level for all distribution parts. On the other hand, the *SuSE* security team is probably one of the best around. Not only do they provide advisories and patches but also additional tools for use. They are also the authors of several advisories for widely used tools like the recent sudo vulnerability.

However, this doesn't mean that other *Linux* distribution developers pay any less attention to security. *Red Hat*, for example, provide a set of security solutions based on their own distribution.

In general, the more popular a distribution, the more likely it is to be explored for vulnerabilities. For example, the Ramen worm was targeted at specific popular distributions.

Of course, security through obscurity doesn't work, and even if virus writers did not test or intentionally target a less popular distribution, there is still a chance that a virus would replicate in that new environment. In fact, using less popular distributions might result in support and administration problems since many applications developers target a limited number of distributions or servers.

There are several so-called 'secure' *Linux* distributions like *Trustix* (http://www.trustix.com/products/tsl/), *SE-Linux* (http://www.nsa.gov/selinux/index.html) and *Immunix* (http://immunix.org/). While they might be an interesting solution for server side, it is not necessarily the best option for a workstation.

Also, consider the problems with applying patches. For example, in *Trustix* you need to disable LIDS before patching can actually take place. While it is a wise idea to use kernel security extensions like LIDS to provide a higher level of security, you still need to consider three factors before choosing patches:

1. Is the security system invisible to the legitimate end user?
2. Is the security system enhancing the security level in the way you want?
3. Is the security system flexible enough to allow users to perform some tasks easily?

The last consideration is very important for developers' machines. For example, developers will need root privileges from time to time. They will also use ptrace for debugging purposes, which is considered insecure, and so on.

For an anti-virus policy we need to consider some additional factors:

- Will the security system allow anti-virus software to run correctly?
- How does the security system affect malware?
- How does the security system affect infection vectors?

For example, an application compiled with *LibSafe* (http://www.research.avayalabs.com/project/libsafe/) or *StackGuard* (*Immunix* OS) should be immune to buffer overflow attacks. This means that worms spreading through exploitation of such vulnerabilities would not be able to infect such a system – at least theoretically. But there are other classes of vulnerability, such as format string bugs (these could also be eliminated by some of the tools provided by *Immunix* and by *LibSafe* for example), race conditions or access to /tmp and proc filesystem (OpenWall). Also, anti-virus software can have problems accessing files while patches like LIDS are activated.

One effective security option is to put every process into a tightly defined cage. This solution is used in Type Enforcement technology incorporated into *SE-Linux*. Similar capabilities are available through Medusa DS9 (http://medusa.fornax.sk/). Such solutions allow the setup of a quite flexible environment for applications. The drawback is that one has to define all the cages for every critical application and this is not as straightforward as it might sound.

The lesson here is simple: first install and configure all network services, then install an anti-virus solution, add security enhancements and configure them. The last step would be to test the setup. This could be done easily with the help of the EICAR test file.

Actually I lied. This was not the last step. After testing the whole setup you need to check everything one more time :)

Rule #1: Install anti-virus software

I am amazed how many *Linux* installations are run without an anti-virus scanner despite the fact that this is one of the fundamental rules for a secure system platform.

Even if we don't consider native *Linux* malware, *Linux* servers are commonly used as email gateways for *Windows* networks. In such cases it would be convenient to stop malware at a gateway level instead of at the LAN level. The use of *Linux* as a quarantine server may be an interesting option for some, since most of today's malware is written for Win32 platforms and cannot be run directly under any Unix platform.

Rule #2: Keep your AV software current

This rule (like the first) is so obvious it shouldn't have to be mentioned, however our experience suggests otherwise.

Rule #3: Check how you are running your AV scanner

For an email gateway the best option would be to run a scanner as a daemon that would somehow interact with your smtp server. In such a case you should consider how to start daemon. Probably the only option is to start it from init/rc scripts, so that the scanner can be loaded during every system start.

Before selecting an anti-virus product test it (or at least ask the developer) to see what the email scanning process looks like. I've seen at least one product that was very aggressive on system resources due to its internal design. Every time a new message arrived a new instance of a scanner was loaded into memory together with the signature database and other required objects. This wasn't a real problem for a few incoming messages every five minutes or so, but for a medium-sized ISP this would be a server killer.

You can also run some system checks and anti-virus software periodically with help of cron or at.

Rule #4: Check for rootkits

You should perform some periodic security checks. There are several commercial and free products that do just that. One of the signs of intrusion is the installation of rootkits. Chkrootkit (http://www.chkrootkit.org/) allows you to check your system locally for signs of rootkit installation and detect some of the most popular rootkits. For more advanced users with Unix system knowledge there is also the wonderful tool lsof, which is capable of showing lists of open files (ftp://vic.cc.purdue.edu/pub/tools/unix/lsof/).

Rule #5: Don't run every service

Most default *Linux* distribution installations start a lot of services. While very few of them are really needed, some of them can lead to intrusion. One of the best examples is lpd daemon which is usually started by default, even though most Internet servers don't have a need for it. When you have no choice but to run a service, check that you are not using a vulnerable version.

Also, don't run every application as root (uid = 0). Many server applications, like *Apache* for example, don't need root privileges (*Apache* runs as a non-root user). Newer versions of *sendmail* (starting from 8.12.0) don't require SUID root attribute on their binary files. Note that basically each new version of *sendmail* has some security fixes so it is wise always to upgrade to the current release.

Unfortunately there are still some poorly written services that run with root privileges when they don't really need to. The most effective solution is to remove them from the system completely. If they perform a critical task for your site it is time to look for a different solution. In any case you have been warned.

Helpful Scripts

There are several scripts and applications that could be quite helpful. One of most popular tools is *Bastille Linux*

(http://www.bastille-linux.org/) which can be run under *Red Hat* (version 7.2 is supported) and *Mandrake*. In addition it can be used with other distributions such as *SuSE* (*SuSE* provides its own set of hardening scripts), *Debian* or *Slackware*, although it might require a bit of work.

Bastille Linux is capable of reconfiguring even non-virgin systems. By setting up packet filtering, tightening file permissions, removing some SUID attributes, disabling C compiler and doing a lot more it proves a good choice for *Linux* servers and workstations. For an end user or administrator *Bastille Linux* is easy to use. It also provides an undo option if something breaks during the process of securing your installation.

Rule #6: Turn packet filtering on

In the short description of *Bastille Linux* I have already mentioned packet filtering. If your *Linux* installation is working in a TCP/IP network environment you should always turn on packet filtering. It allows you to manage access to the network services provided by your workstation. This is very helpful when you need to run services like printer daemon (lpd) that shouldn't be available publicly.

It doesn't matter whether you are sitting behind a firewall already – there is always risk of internal LAN infection. In such cases worms could spread very quickly unless network services are protected.

Conclusion

Protecting *Linux* hosts from being infected is not necessarily a trivial task, but by using some basic security rules *Linux* can be made into a fairly secure platform.

Unfortunately some of the security extensions or mechanisms available to the *Linux* community weren't designed with worms or viruses in mind. In turn, some kernel patches can make anti-virus software unusable. From my personal experience I have noticed that many *Linux* administrators don't treat malware as a danger. On the other hand, *Windows* administrators tend to forget that *Linux* workstations can be (and are) network servers, running *sendmail* for example.

Some of the kernel patches or tools mentioned can work only on x86 platforms. While some script kiddies' tools are also only able to run on x86 systems, this does not make our system more secure. Writing shellcode for architectures other than x86 is not only possible, but it is done on a daily basis.

Finally, you should note that the *availability* of many security features and extensions (some of them are not available to Win32 systems) doesn't make your system secure or invulnerable. First you have to deploy them, and then use them wisely.

OPINON

The Easy Out, the Quick Fix, the Silver Bullet and the Big Idea

David Perry
Trend Micro, USA

There is a theme that pervades our best fairy tales: the magic fix for the untenable situation. Under attack from vampires? Get a stake and some garlic. Werewolves the problem? You need a silver bullet. Being held by an evil magic dwarf? His name is Rumpelstiltskin.

Great Expectations

The idea of the magic fix is not confined only to our stories, but exists also in our expectations of the real world.

When confronted by a sobering diagnosis from the doctor, when facing personal disappointment or tragedy, even when confounded by a world recession or the horror of war, we want the easy way out.

Some of us may progress to greater levels of understanding and maturity in the world, but isn't it always there? That silver bullet is always the first idea to show up and the last to loosen its hold.

I want to be clear that I am not talking about religion. As near as I can tell, a spiritual path is as far from an easy answer as the deepest skepticism. The magic fix, on the other hand, is nothing if not easy.

In the world of computer virus research and the AV industry, we are no different, and no better than the world of people at large. Looking back over the last decade, we can identify dozens of big ideas, all promising to end this virus thing *once and for all!*

The Magical Belief in the Next OS

We have heard this one time and again. At the beginning of the *Windows 95* era, I remember hearing that protected mode would block any boot sector virus, preventing it from reinfecting.

Although this is true as far as it goes, it does not hold anywhere near to a complete model of the world. One major AV company was ready to 'sunset' its AV efforts based mainly on this premise.

Some Other OS to the Rescue

This is an idea I hear constantly – that *Linux* (or *BSD*, or Apple system IX or what have you) is so much better

engineered than other operating systems, and that viruses on *this* OS will be 'impossible'. If you are looking for a virus-free operating system, I suggest you use paper and a pencil.

Heuristics

I am a big fan of heuristic detection, but we have to watch the use of this word when speaking to the press! I suspect that many reporters believe that heuristics is something that comes in a five-gallon can, and can be applied at will with magical effect. Of course, a specific rule or detection trick will detect only the sort of thing coherent with its base model.

Even a cursory look at the history of computer viruses (let alone the greater mish-mash of history in general) will show that, on many occasions, we have had to hammer the walls out of our definitions just to keep our taxonomy intact. If you are drafting a letter to tell me that your personal internal logic has always been perfect and consistent, get professional help! (This is called *denial*!)

Integrity Checking

I am also a big fan of integrity checking. Unfortunately, this works only on something with assumed integrity.

Integrity checking is great for parasitic infections of all types, but it misses many types of virus, Trojan and worm that are bereft of parasitic action. What amazes me is that so many companies have come to conquer with this the only arrow in their quiver.

Deterrence by Punishment

This idea comes up frequently in the mainstream press and with customers: 'If we could just catch and punish some of these virus writers …'

It is obvious that vandalism (if you will excuse my classification of the act of virus writing and distribution as a form of vandalism) has been a characteristic part of human activity throughout history, springing from a variety of root causes and taking a variety of forms.

Furthermore, vandalism has yet to be deterred by any threat of punishment (although it is seen to be greatly reduced in societies of lower density and in societies of draconian social control). However, the questions concerning deterrence and punishment are too great to be contemplated in this piece.

The Magic Box

I shall name no names, but there have been, from all quarters, efforts to describe a certain commercial product as *the* ideal anti-virus solution. With (Product X) installed (we are told), no virus can get through. When something does get through, the skirts of the definition of virus are rearranged to exclude the offending intrusion. From the look of things as they currently stand, it seems that the Red Queen was right when she said to Alice, 'Now, here, you see, it takes all the running you can do to keep in the same place.'

Taking Stock

I could go on. It is evident to those of us who work in these trenches that both the complexity of code bases and the availability of open 'doors' in both operating systems and application code will, in time, increase both the number and type of exploits. Already I accept that today's best paradigm for a computer virus will (all things proceeding logically) be surpassed in orders of magnitude by the code we will see in the malware offerings of tomorrow.

Computer viruses (in this case I mean viruses in the broadest sense, throw in the whole bestiary) are not one thing. They are neither all aimed at the same sort of system, nor aimed at the same sort of user. They are neither all created for the same purpose, nor by people with the same world view. Frequently I am reminded that my own analysis is constantly seeking a single model to just fit the thing *metaphorically.*

We now live in a world with somewhere between 60,000 and 100,000 strains of virus (depending on who's counting), of which only about a thousand have ever infected anyone (ditto). There are vast numbers of viruses displaying clever methodology that never really caught on (e.g. Dir2, commander bomber, *ad nauseam*). And there are almost a dozen virus types, one or two of which have been synonymous with the word virus even to the engineers inside the AV companies and, I suspect, to the researchers inside our best temples of learning.

That Unrelenting Silver Bullet

This is a complex and constantly shifting landscape. To sit a layman down and bring him up to speed even on the basic vocabulary can take hours and it can take days to give him a view of the real thing. We have all had to give up some of our cherished axioms and overviews as exception after exception has muddied the waters. And yet we see offered to us every day a single model, an instant fix, a *silver bullet.*

COMPARATIVE REVIEW

Making an Entrance: SuSE Linux

Matt Ham

As *Virus Bulletin*'s first *Linux* comparative, this review was embarked upon with some trepidation. In general, *VB* comparative tests have become easier to carry out as time has progressed and the obscure foibles of the various products have made themselves known. Without the benefit of this background knowledge, it was anyone's guess as to what the products in this test would present by way of pitfalls.

In addition, there is an array of testing tools available to ease the process of comparative tests, as well as various scripts and utilities, all of which are *Windows*-based and of no use in a *Linux* test.

With such a show of anxiety at the start, I shall break with tradition and state that all products proved testable for on-demand detection, though the methods used to produce these results differed slightly from those usually employed in *VB* comparative tests.

Of the eleven products submitted for testing, only three offered on-access scanning located entirely upon the *Linux* server, and the results from the testing of these modules were less than impressive. This being the case, the results of on-access scanning tests are bundled together after the main body of the review.

Several of the product lines that are regular contenders in *VB*'s comparatives are absent from this review – either due to their being at beta stage or because this platform is not supported by their manufacturers. Furthermore, a sizeable proportion of the products reviewed are scheduled for major upgrades in the near future – the *Linux* anti-virus market is still young and subject to change.

Test Sets

The test sets for this comparative review were based upon the standard *Virus Bulletin* comparative test sets. The In the Wild (ItW) set was aligned to the *WildList Organization*'s February 2002 WildList.

In addition to the usual contents of the test sets a number of *Linux* worms and viruses were added. These fall into two categories: worms transferred as archives after an initial exploit has given local access rights and ELF file infecting viruses. As yet, the number of these is not great, but more files will be added with future test set updates.

Other additions to the test sets included two viruses in the polymorphic test set, W32/CTX and W32/Fosforo. Again, the polymorphic test sets can be expected to have several further additions in the near future.

Of the additions to the ItW test set one is more noteworthy than its impact in the real world might have suggested. W32/Heidia.A is a .ZIP file infector which relies upon manual running to insert itself into existing .ZIP archives. The main code for this process was the file included in the ItW set – though a pair of infected .ZIP files were added to the standard set. The addition of archives such as these will instantly strike a detection rate rift between those scanners which look inside archives by default and those which do not.

This difference will be made all the more apparent by the presence of the *Linux* worms. *Linux* worms are commonly transferred as archives of files – and, clearly, these will not be scanned (at all) on-demand by products which do not consider archives worth scanning.

In the cases of Lion.A, Ramen and Adore, the files placed into the test set consisted of the contents of the archive as well as the archive itself. This left Lion.B and Lion.C which were represented only by their archived form. Another likely problem file is Cheese, which is UUE encoded.

For speed testing the standard clean sets were used – though with another *Linux*-specific addition. In order to test the rate of scanning for native *Linux* files the contents of /bin, /opt and /sbin were selected as a further test set. Since these files may be subject to replacements or additions when software is installed, a copy was made of these two directories. Testing was performed on this copy so as to ensure that each product was scanning an identical test set.

Test Procedure

All test sets were stored in RAR archives or compressed machine images and restored between tests.

On-demand tests were performed locally, with the bulk of the test sets being scanned while located on FAT partition. The exception to this was the *Linux*-specific malware which was scanned while located in a directory in the root of the *Linux* installation. This was so that the files would be scanned on their native partition format.

A cursory inspection of the first few products to arrive suggested that the most reliable method of detection in this test would be to standardize on detection by deletion of infected files. Primarily, this was because this is a far quicker process than sifting through the results generated by programs which do not support logging except by redirection of STDOUT to a file. Deletion proved to be a good solution (except in those cases noted in the individual product comments).

On-demand tests	ItW File		Macro		Polymorphic		Standard		Linux	
	Number missed	%	Number missed	%	Number missed	%	Number missed	%	Number missed	%
CA Vet Rescue	0	100.00%	0	100.00%	460	97.00%	1	99.94%	64	51.31%
Command AntiVirus	0	100.00%	0	100.00%	486	93.01%	4	99.79%	78	52.20%
DialogueScience DrWeb	0	100.00%	0	100.00%	399	99.14%	1	99.98%	35	81.91%
Eset NOD32	0	100.00%	0	100.00%	454	97.75%	0	100.00%	77	39.81%
FRISK F-Prot	1	99.92%	0	100.00%	399	99.14%	1	99.98%	35	81.91%
GeCAD RAV	1	99.92%	0	100.00%	411	96.79%	19	99.22%	42	68.43%
Kaspersky KAV	0	100.00%	0	100.00%	399	99.08%	0	100.00%	10	92.41%
NAI VirusScan	0	100.00%	0	100.00%	413	98.78%	2	99.87%	24	77.80%
Norman Virus Control	3	99.70%	18	99.68%	473	93.76%	13	99.49%	19	84.72%
Sophos SWEEP	0	100.00%	5	99.87%	476	93.31%	18	99.43%	54	58.90%
VirusBuster VirusBuster	1	99.95%	0	100.00%	493	91.01%	14	99.55%	7	90.00%

In order to test on-access scanning, the *Linux* server was connected by *SAMBA* to a *Windows 2000 Professional* workstation. From here, the standard *VB* test tools were used to move recursively through the test set, opening each file in turn so as to trigger on-access scanners.

Finally, the matter of testing the speed of scanning was addressed. Again, the standard *VB* clean sets were selected for scanning on a *Windows* partition situated locally, while a *Linux* test set was constructed – consisting, in this preliminary incarnation, of the contents of the /sbin, /bin and /opt directory trees of the test *Linux* machine. Since several of the products install within the /opt tree, these files were copied into a dedicated test directory rather than being scanned *in situ*.

Computer Associates Vet Rescue 10.5.0.0

ItW	100.00%	Macro	100.00%
Polymorphic	97.00%	Standard	99.94%

CA Vet Rescue displayed several odd quirks, not all of which were unique to this product, but since it is first alphabetically this seems an appropriate place to discuss these oddities.

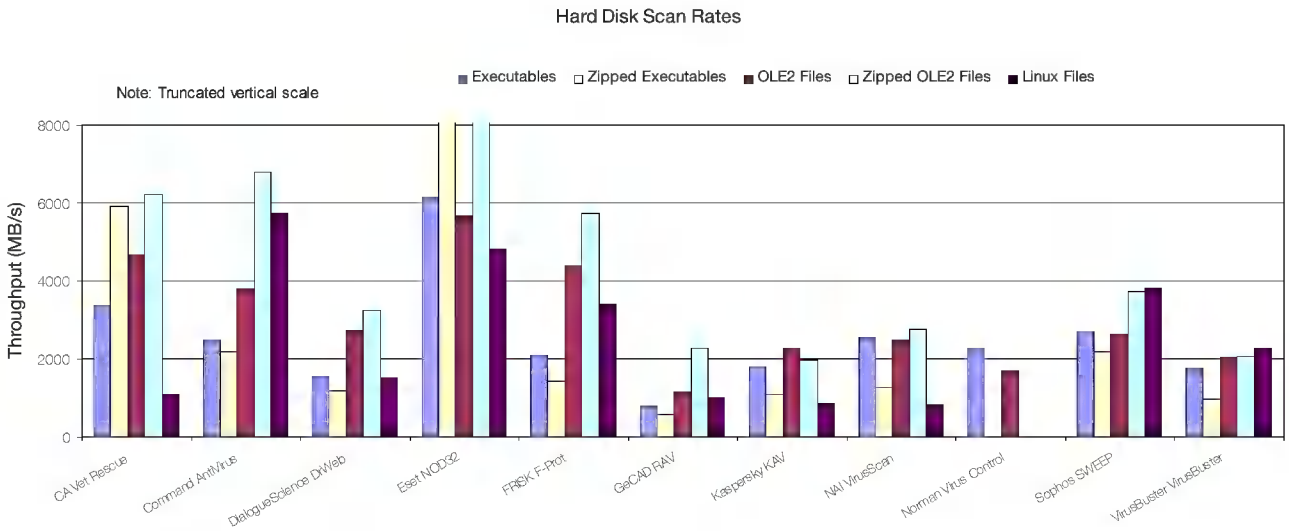
The most commonly encountered problem was that of accepted command line arguments. Using the –? argument for help produces a brief list of arguments followed by a more detailed description of what each of these does. This is all well and good, except that in many of the products, *Vet* included, the two lists of acceptable arguments do not tally. In other products the two lists tally, yet do not agree with the usable options – an even more confusing situation.

Vet Rescue hints as to the source of these unhelpful proceedings, since it announces itself as 'rescue.exe'. This leads to the conclusion that the command line argument handling code and other associated routines have been considered as being machine-portable from the DOS command line scanner. This may be true from a purely code-based point of view, but it would have been preferable for the text to have been taken into consideration when this portability issue was decided upon.

The lack of available command line options in *Vet Rescue* is quite marked, but certainly not unique to *Vet*. *Vet* was unusual, however, in requiring the target directory to be included before any options in the command line – which is opposite to the *de facto* standard. The lack of functionality may well explain *Vet Rescue*'s impressive scanning speeds.

A full tally of ItW and macro detections bode well for *Vet*'s fortunes, but results slipped slightly away from perfection on the polymorphic and standard sets, while the *Linux* set saw a detection rate of only a little over 50 per cent. However, *Vet*'s *Linux* detection rate proved to be not far below the average detection rate managed by products in this set.

It should be noted in relation to the *Linux* sample detection rates that, with such a small sample set as that used for these tests, there is great scope for errors in estimating the detection ability of a product. Until the number of samples in the set has increased significantly, no great messages



should be inferred from these figures – which are provided here for interest.

Command Software AntiVirus 4.64.0

ItW	100.00%	Macro	100.00%
Polymorphic	93.01%	Standard	99.79%

Following in the footsteps of *Vet*, *Command AntiVirus* demonstrated some odd behaviour. In this case it was a point blank refusal to delete any file which potentially could contain useful data – notably archives and OLE files. Since quarantining of these files was not permitted either, another method of deletion was selected.

The product was permitted to disinfect the samples which it refused to delete, and those files with changed checksums were deleted as having been declared dirty. As a sanity check the checksumming was performed without disinfection – to guard against the remote possibility that the scanner would alter checksums in some arcane manner. The scan with no disinfection showed no change in checksum – as would be hoped.

After obtaining results in this way the detection rates were certainly not disappointing at first glance although, admittedly, the *Linux* samples were discovered with only 50 per cent regularity and there were a number of misses in the polymorphic test set.

In the polymorphic set, the newly-added Win32/Fosforo samples caused problems – and the slightly older W32/Zmist.D samples evaded detection completely. However, a more concerning set of missed files was hidden behind the façade of full detection In the Wild.

The newly In the Wild virus W32/CTX is represented by ten samples in the ItW test set and, by reason of its polymorphic nature, is represented in the polymorphic sample set too, with 84 further samples. All ItW samples were detected, but 15 of the samples in the polymorphic set evaded detection. Such imperfect detection is not uncommon with complex polymorphics – but is undesirable nevertheless.

DialogueScience DrWeb 4.27a

ItW	100.00%	Macro	100.00%
Polymorphic	99.14%	Standard	99.98%

DrWeb registered its standard tally of suspicious files during the tests on the clean set, though there were surprises in store elsewhere.

I will admit that the detection of a virus In the Wild (in this case W97M/Pecas.B) using heuristics is not shocking. *DrWeb*, however, has a good record of identifying infected files accurately and exactly, so it was mildly surprising that on this occasion it detected only heuristically.

Other than this unexpected change, detection rates were good, though lowered by the influx of *Linux* and polymorphic viruses, which have added a significant new challenge to the companies submitting to this comparative. However, *DrWeb* was less affected by the new samples than many of the other products on test.

As with some of the other products there was no obvious method of determining a version number for the product, other than using the version number provided as the name and description of the installation RPM. There was also a slight difficulty in persuading *DrWeb* to delete what it considered to be archive files – though here these were only *PowerPoint* and some VBS files.

Eset NOD32 1.990

ItW	100.00%	Macro	100.00%
Polymorphic	97.75%	Standard	100.00%

Hard Disk Scan Rate	Executables			OLE Files			Zipped Executables		Zipped OLE Files		Linux Files		
	Time (s)	Throughput (MB/s)	FPs [susp]	Time(s)	Throughput (MB/s)	FPs [susp]	Time (s)	Throughput (MB/s)	Time(s)	Throughput (MB/s)	Time(s)	Throughput (MB/s)	FPs [susp]
CA Vet Rescue	162.0	3376.1		17.0	4666.7		27.0	5904.3	12.0	6217.3	169.0	1082.5	
Command AntiVirus	221.0	2474.8		21.0	3777.8		73.0	2183.8	11.0	6782.5	32.0	5717.0	
DialogueScience DrWeb	354.0	1545.0	[16]	29.0	2735.6		136.0	1172.2	23.0	3243.8	120.0	1524.5	
Eset NOD32	89.0	6145.3		14.0	5666.7		17.0	32172.5	3.0	26444.6	38.0	4814.3	
FRISK F-Prot	263.0	2079.6		18.0	4407.4		111.0	1436.2	13.0	5739.0	54.0	3387.9	
GeCAD RAV	690.0	792.7	[1]	69.0	1149.8		277.0	575.5	33.0	2260.8	181.0	1010.7	
Kaspersky KAV	307.0	1781.5	[18]	35.0	2266.7		147.0	1084.5	38.0	1963.4	219.0	835.4	
NAI VirusScan	216.0	2532.1		32.0	2479.2		124.0	1285.6	27.0	2763.2	224.0	816.7	
Norman Virus Control	241.0	2269.4		47.0	1688.0	[77]	n/a	n/a	n/a	n/a	n/a	n/a	
Sophos SWEEP	202.0	2707.6		30.0	2644.5		73.0	2183.8	20.0	3730.4	48.0	3811.3	
VirusBuster VirusBuster	310.0	1764.3		39.0	2034.2	[1]	166.0	960.3	36.0	2072.4	80.0	2286.8	

Once again, *NOD32* was significantly speedier than any of the other products on test, and it maintained its excellent detection rate on the old favourites in the *VB* test sets.

However, there proved a good deal more to challenge *NOD32* than usual, partially on account of the additions in the *Linux* test set. Scoring the lowest percentage of any scanner when faced by ELF format viruses, there is room for improvement for *Eset* here. Similarly, a number of the newly-added W32/Fosforo samples were missed by *NOD32*, resulting in the largest number of misses for *Eset*'s product for many comparatives.

Frisk F-Prot Antivirus 3.11

ItW	99.92%	Macro	100.00%
Polymorphic	99.14%	Standard	99.98%

The *F-Prot* product suffers from similar command line argument oddities to its close relative *Command AntiVirus*. At least in this case the problem is noted in the documentation. The documentation is also quite clear in stating that this is a product which is still under development, with several possible new avenues opening up to it in the near future.

As befits a product using the same engine, the results of the detection tests for *Command AntiVirus* and *Frisk F-Prot* were very similar. This similarity went as far as identical results in all but the ItW test set. Here, *Frisk* missed the .EML-extensioned sample of W32/Nimda.A – presumably .EML format files are excluded from scanning in order to reduce scan time.

This raises an intriguing problem as far as scanning from or upon a *Linux* machine is concerned. Many products still employ extension lists as a first filter when determining which files are to be scanned. It is not uncommon for scanners to check for executable content disguised by extension, but this is by no means universal.

On a *Linux* machine, however, extensions are essentially meaningless in many cases, and are more likely to be descriptive than any guide as to whether the file in question is an executable.

This is not so much a problem for products which can perform intelligent file-typing – but it may be irritating to developers who have traditionally relied upon extensions as an easy way of avoiding processor usage.

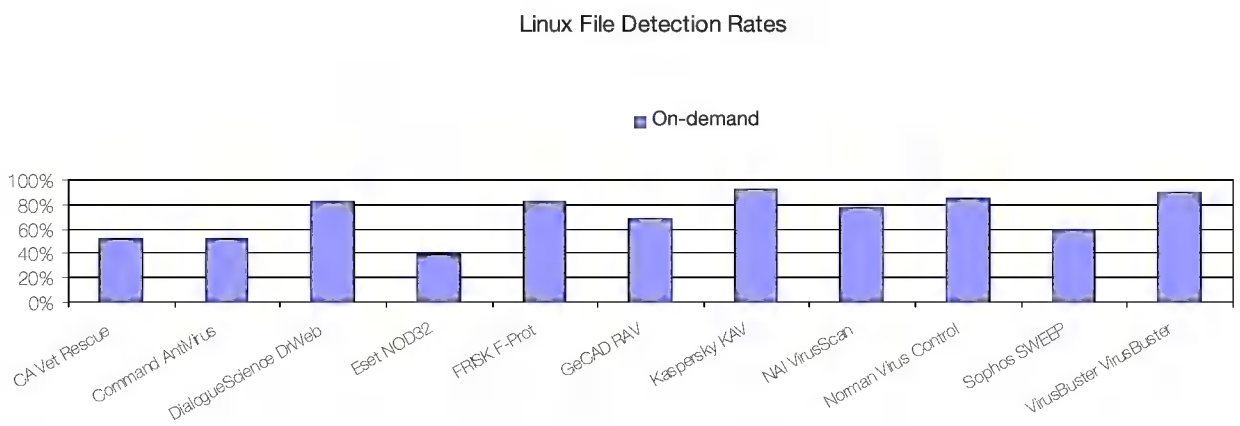
GeCAD RAV AntiVirus 8.5

ItW	99.92%	Macro	100.00%
Polymorphic	96.79%	Standard	99.22%

RAV was the only product in the test to boast a graphical interface – though the command line version was used for testing.

As befits a product which has seen more development than most, the command line options within *RAV* were numerous and bore more resemblance to the feature set usually seen on a DOS scanner.

Since the *RAV* scanner has been the subject of a recent standalone review, discussion of features here can be skimmed past speedily. However, this feature set did not protect against accidents, and after creditable detection rates in most categories, a miss due to the .EML version of W32/Nimda.A In the Wild will, no doubt, be galling for *GeCAD*.



Kaspersky AntiVirus 4.0.0.1

ItW	100.00%	Macro	100.00%
Polymorphic	99.08%	Standard	100.00%

One of the important differences to keep in mind when returning to a Unix-based operating system from *Windows* is the need for correct capitalization, a feature which led to some problems with the *Kaspersky* product.

Irritatingly, the archives provided for updating the product were all fully capitalized, whereas the program expects file naming in lower case lettering. This led to the somewhat tedious need to rename all of the definition files supplied, of which there were a large number, each dedicated to a certain type of threat.

In a rather idiosyncratic display, *KAV* defaulted to disinfecting files within archives on several occasions – despite being explicitly configured to perform deletions.

When this had been worked around, however, *KAV*'s performance was very much a return to form after some unlucky outings in recent *VB* comparative reviews. Certainly at the top of the detection range as far as the *Linux* files were concerned, *KAV* showed good detection all round.

When scanning the clean test set there was a moment of interest, as several possible false alarms appeared where none have been seen recently. The question was raised as to whether these should be classified as false alarms or merely as suspicious files. The announcement of some feeble joke program as 'VIRUS-noseless-dog-joke' has been a constant irritation to testers and end users alike – and in this case *KAV*'s alerts proved to be false alarms triggered by the detection of some form of greetings card.

Gratifyingly, however, the messages produced were as clear as might be hoped in the circumstances – declaring that what had been found was 'not-a-virus;GreetingCard.SLR'. With such a label, what had initially been considered a possible false alarm was speedily downgraded to merely a suspicious file.

NAI VirusScan 4.16 4188

ItW	100.00%	Macro	100.00%
Polymorphic	98.78%	Standard	99.87%

It was, perhaps, a little surprising that *NAI*'s product did not arrive as a fire-and-forget RPM package, but in the more humble guise of a gzipped tarball. Far from being typical from a company which has in the recent past indulged in the home user feature race with large competitors, this return to simplicity was reminiscent of the earlier days of *NAI*'s ancestral companies.

An irritating if not fatal niggle was that the default settings were not listed when command line switches were displayed, which left a large number of possibly irrelevant selections being used to avoid unwanted disinfection and the like.

Similarly, as noted for other products, the treatment of documents as archives makes it difficult to delete these directly. The fact that this proved to be a constant problem in this *Linux* comparative, while not having been an issue when dealing with any other platform, does seem odd.

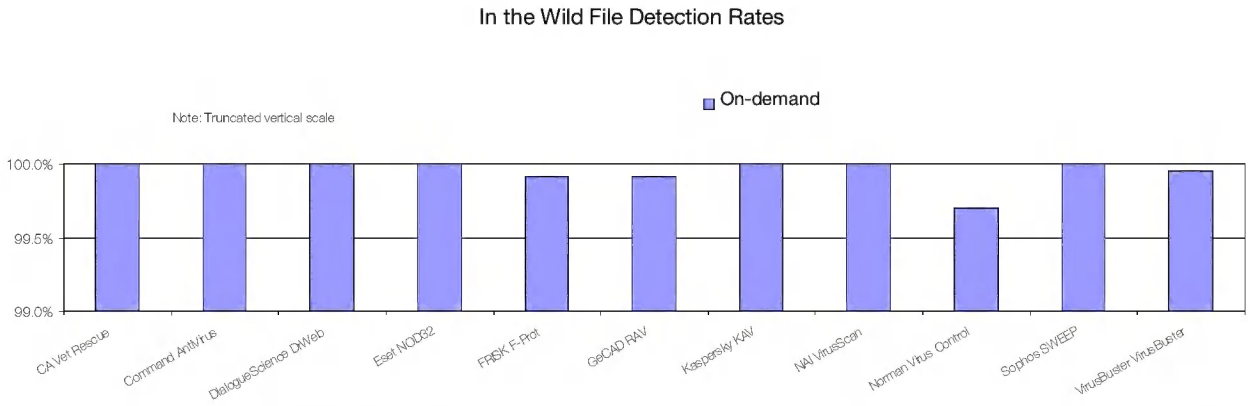
As far as misses in the detection tests are concerned, *VirusScan* was another product where a streak of bad luck seems finally to have come to an end. Detection was certainly at a better level than has been the case lately – and only in the *Linux* set can any weaknesses be identified.

After the good news there remains one fly in the anti-viral ointment, this being *VirusScan*'s speed of scanning. This was in the slower half of the field where the age-old *VB* clean sets were concerned, and the slowest of all those tested when *Linux* clean files were scanned.

Norman Virus Control 5.3-1

ItW	99.70%	Macro	99.68%
Polymorphic	93.76%	Standard	99.49%

Norman's product scored highly where the provision of reports was concerned – which is odd indeed, since this is



not a feature that is supported directly in those versions tested on other platforms. Especially appreciated was the list of clean files – this may be of somewhat limited use to the end user, but is excellent for a reviewer.

The version of *NVC* supplied seemed to encounter numerous difficulties when faced with the *VB* clean test sets. On non-archived Win32 and OLE2 files all was well, but on scanning the .ZIP test sets and the *Linux* test set, which includes some archives, the program ground to a halt – not before producing some cryptic error messages and a few random characters on the screen.

Equally disturbing was the program's behaviour when scanning the OLE2 files. The increase to 77 suspicious files detected in this test set must be indicative of an error somewhere.

Again, when pure detection was inspected *NVC* demonstrated some unexpected behaviour. This manifested itself in the missing of files ItW which have been detected by *NVC* on other platforms since time immemorial.

Whether the problems encountered here are specific to the flavour of *Linux* on test, or they are more general in nature, it can only be hoped that they will be banished in short order.

Sophos SWEEP 3.55

ItW	100.00%	Macro	99.87%
Polymorphic	93.31%	Standard	99.43%

Like roughly half the packages submitted, *SWEEP* arrived as an archive rather than as an RPM package – though an installation shell script was supplied to ease matters. The script requires that a *SWEEP* user and group are set up, though it seems that these are not used unless the machine is to become an *InterCheck* server.

In terms of detection, like *Norman*, *SWEEP* was hit fairly hard by the addition of W32/Fosforo to the polymorphic test set, as well as the numerous new archive files which were added to the test sets this month.

Two other features of note came to light in this review. The first was that the IDE files used to add virus detection to the product must be placed manually in a directory which is not the main program directory – slightly counter-intuitive.

Perhaps of greater note is that the detection for W32/CTX was added on the day of the review deadline – though the virus had been declared to be In the Wild for some time by that point.

VirusBuster VirusBuster 1.06

ItW	99.95%	Macro	100.00%
Polymorphic	91.01%	Standard	99.55%

The main frustration with *VirusBuster* came when trying to determine a version number for the product – this seemed impossible to determine from within the software. In the end the package version number was selected – though quite how a user will be able to tell which virus definitions are loaded remains a mystery.

On the detection front *VirusBuster*'s behaviour is best described as variable. Detection is good in all areas, with the *Linux* detection rates being in the top of the field, but the polymorphic detection rate is distinctly weak.

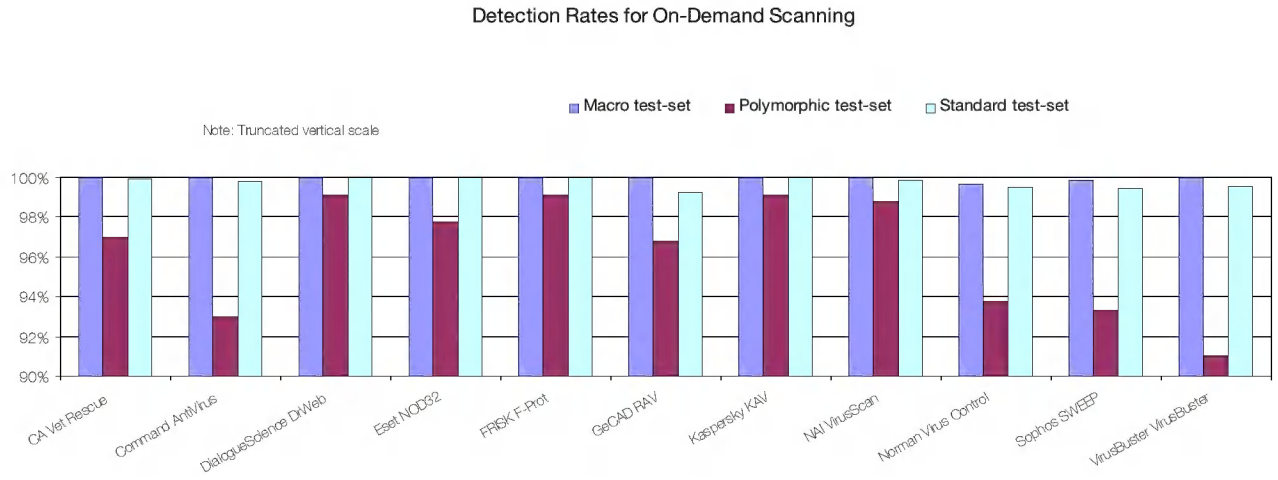
In the past there have been complaints that too many products detect almost all files in the test sets. Frequent additions to the polymorphic set should mean this will edge well away from a collection which can be detected fully.

On-Access Scanning

Of the products reviewed four had some form of on-access component.

The first to be considered outside the scope of the review was *Sophos SWEEP*. Although *Linux* can be used to support an *InterCheck Server*, which supports on-access scanning, there is no *InterCheck Client* for *Linux*.

In effect, this means that on-access scanning can be done by a *Linux* machine – but can only be performed on behalf of a



machine which offers support for the *InterCheck* client, thus making the machine incapable of scanning itself.

The three remaining products to offer on-access scanning offer this feature as a kernel module. This allows for fully native file access interception – but such modules are kernel-dependent, which leads to problems in that a standardized module cannot be supplied.

Kaspersky Lab circumvents this problem by supplying make files and source for the module, which is compiled by the user. Unfortunately, on the default installation of *SuSE Linux* used for testing, compilation failed to complete.

The situation for *DialogueScience*'s *Dr Web* was somewhat different, in that *DialogueScience* supplied a pre-constructed module which was tailored to the kernel versions under test. Again, there was one fatal problem with this, in that the version of *SAMBA* used in this test was not compatible with *Spider*'s requirements.

The most hopeful performance was offered by *ESET*'s *Amon* module – which loaded and performed interception as advertised when test accesses were performed on individual files. Admittedly, the behaviour was not particularly informative to the user, since access was denied to infected objects without any explanation.

With such a promising start it came as something of a disappointment when the on-access tests were commenced.

On numerous occasions during the on-access scanning tests the *Linux* machine simply locked up – accepting no input whatsoever other than the power switch. Again, testing was left for standalones, where experimentation is a luxury not possible in the time available for a comparative.

VB 100% Awards

All this talk of on-access scanning steers the course of discussion to that old favourite, the VB 100% awards.

The expectation that a product should be able to detect both on access and on demand remains a primary feature in the awarding of the VB 100% logo.

As indicated, there were no products that were able to install upon the stated default test machine network and thus none in this comparative was eligible for the VB 100% award.

There is no denying that there are great problems for the developers in achieving portable code for a multiplicity of kernels, and these may well prove insurmountable for those users who make use of particularly mephistophelean kernel configurations.

It is equally clear, however, that the on-access components have worked on those kernels that are in more common usage. The challenge for obtaining a VB 100% award in future tests will be partially in providing such a component – but more in providing one which will work on a wide range of platforms.

More than ever this means that the *Linux* comparatives cannot be seen as a representation of anything other than how the selected test configuration is supported. Making the assumption that these results would be identical on other kernels or configurations would be foolhardy.

Conclusion

The addition of *Linux* as a platform for comparative review has certainly brought some new and challenging problems to the testing process, due simply to the smaller number of features that can be taken for granted on this platform.

Anti-virus products are still, by and large, quite young in the *Linux* market, with those features such as quarantining, which are taken for granted elsewhere, being a rarity in the products reviewed.

It does look as if a certain degree of market impetus is present, if the rapid changes in the products available and the features on existing products are anything to judge by.

One disappointment, however, was the generally poor level of detection for the *Linux* files which were added into the test sets.

Of course, some of these *Linux* files are certain to be missed without the use of archive scanning (though this could prove a good reason for enabling the scanning of archives by default, at least on this platform).

There is something of a potential problem involving circular reasoning with this lack of detection. The nature of *Linux* is such that the need for virus protection on this platform is somewhat lower than it is on other platforms – providing the correct procedures are followed. For this reason, the development of anti-virus products for *Linux* has been slow historically.

However, if the rate of detection of *Linux* files is low, few customers are likely to come forward, there will be no impetus for development and detection rates are unlikely to increase.

Whether this cycle is realized or boom ensues only time will tell.

Technical Details

As this is the first in a potentially long series of *Linux* comparative reviews, the technical details come with what amounts to an explanatory note.

The version of *Linux* chosen was selected deliberately so as not to be one of the most commonly installed, while still being sufficiently large to have relevance to developers. *SuSE* version 7.2 was chosen over version 7.3 as this was considered to be the more stable of the two.

In effect, the ideal platform for the test would provide a slight challenge to a product's cross-platform abilities, yet at the same time avoiding any unnecessary obstacles from known bugs.

The test environment, as noted above, was designed to mimic at least a possible real-world situation. In this case a *Linux* machine using *SAMBA* was considered a 'normal' application, while not the simplest for an on-access scanner to negotiate.

Technical Details

Linux machine: 750 MHz AMD Duron workstation with 128MB RAM, 8 GB and 4 GB dual hard disks, CD-ROM, LS120 and 3.5-inch floppy, running *SuSE Linux 7.2* (Glibc 2.2, *Linux* kernel 2.4.4)

Client machine: 750 MHz AMD Duron workstation with 128MB RAM, 8 GB and 4 GB dual hard disks, CD-ROM, LS120 and 3.5-inch floppy, running *Microsoft Windows 2000 Professional*.

Connected by *Samba 2.2.0 -15*.

Virus test sets: Complete listings of the test sets used are at http://www.virusbtn.com/Comparatives/Linux/2002/02test_sets.html. A full description of the results calculations protocol is at http://www.virusbtn.com/Comparatives/Win95/199801/protocol.html.

SUBSCRIPTION RATES

Subscription price for 1 year (12 issues) including first-class/airmail delivery:

UK £195, Europe £225, International £245 (US$395)

Editorial enquiries, subscription enquiries, orders and payments:

Virus Bulletin Ltd, The Pentagon, Abingdon Science Park, Abingdon, Oxfordshire, OX14 3YP, England

Tel 01235 555139, International Tel +44 1235 555139
Fax 01235 531889, International Fax +44 1235 531889
Email: editorial@virusbtn.com
World Wide Web: http://www.virusbtn.com/

US subscriptions only:

VB, 6 Kimball Lane, Suite 400, Lynnfield, MA 01940, USA

Tel (781) 9731266, Fax (781) 9731267

END NOTES AND NEWS

Information Security World Asia 2002 runs from 16–18 April 2002 in Singapore. For further information about the show or to register online visit http://www.isec-worldwide.com/isec_asia2002/.

Infosecurity Europe 2002 takes place 23–25 April 2002 at London's Grand Hall, Olympia. For further details and pre-registration visit http://www.infosec.co.uk/.

The Southwest CyberTerrorism Summit will be held 4 May 2002 in Dallas, TX, USA. Topics include wireless hacking, cyber-attacks, information warfare, privacy, computer viruses, industrial espionage and identity theft. For more information about the event visit http://www.DallasCon.com/.

Infosec 2002 takes place 28–30 May 2002 at CNIT, Paris La Défense, France. This three-day event will run concurrently with SIMBIOM, the First International Biometry Exhibition. For more information, including an exhibitor list and details of the conference and tutorials, visit http://mci-salons.fr/infosec/.

The RSA Conference 2002 Japan runs 29–30 May 2002 at the Alaska Prince Hotel, Tokyo, Japan. Nine tracks have been named for the conference, including: network threats and security, PKI, government policies and rules, advanced technology, risk management and corporate security and new products and technology. More information can be found at http://www.rsaconference.com/.

The 11th Annual EICAR Conference and 3rd European Anti-Malware Forum takes place 8–11 June 2002 in Berlin, Germany. For further deatils and to register for the conference see http://www.eicar.org/.

Papers and presentations are now being accepted for the Black Hat Briefings 2002 conference. The conference is to be held from 31 July to 1 August 2002 at the Caesar's Palace Hotel in Las Vegas, USA. Submissions must be received by 1 May 2002. For further details of the conference see http://www.blackhat.com/.

Information Security World Australasia 2002 will be held 19–21 August, 2002 in Sydney, Australia. The conference and exhibition represent the region's largest dedicated IT security show. For full details see http://www.informationsecurityworld.com/.

The 9th International Computer Security Symposium, COSAC 2002, takes place 8–12 September 2002 at Killashee Hotel, County Kildare, Ireland. Cost of registration includes your choice of 40 symposium sessions, five full-day master classes, and the COSAC International Peer Group meeting, in addition to full-board accommodation and meals. Register at http://www.cosac.net/.

The 12th International Virus Bulletin Conference will take place at the Hyatt Regency, New Orleans, LA, USA from 26–27 September 2002. Watch out for the full programme details at http://www.virusbtn.com/.

Information Security Systems Europe 2002 will be held in Disneyland, Paris, from 2–4 October 2002. For more information visit http://www.isse.org/.

The Third Annual RSA Conference 2002, Europe is to take place 7–10 October 2002 at Le Palais Des Congrès de Paris, France. As well as keynote presentations there will be more than 85 individual breakout sessions on topics ranging from enterprise security to hacking and intrusion forensics. See http://www.rsaconference.com/.

The March 2002 WildList will be the last 'until further notice'. *WildList Organization* Chief Executive Shane Coursen has announced that he is seeking full-time employment in the AV community as an anti-virus researcher, consequently the compilation and distribution of the WildList will be put on hold for the time being.

Panda Software* is to launch *Panda Antivirus Enterprise Suite, a new anti-virus solution for corporate networks. The six modules included in the Suite can be purchased separately or as a complete package. See http://www.pandasoftware.com/.

The **Securities & Exchange Commission (SEC) has embarked upon a 'Formal Order of Private Investigation' into the accounting practices of *Network Associates, Inc.*** during the 2000 fiscal year. It is believed that the inquiry relates to accounting issues that predate the current management team's arrival in early 2001. *NAI* will postpone to a later date its plans to launch an exchange offer for all outstanding publicly held shares of Class A common stock of *McAfee.com*. For more information see http://www.sec.gov and http://www.nai.com/.